ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КАФЕДРА ВС

Лабораторная №4

По дисциплине «Архитектура вычислительных систем»

«Оптимизация доступа к памяти»

Выполнил: студент гр. ИП-013

Копытина Т.А.

Проверил: ассистент кафедры ВС

Насонова А.О.

Новосибирск 2022г.

Оглавление

ПОСТАНОВКА ЗАДАЧИ 3
ОПИСАНИЕ ПРОГРАММЫ 4
РЕЗУЛЬТАТ ПРОГРАММЫ 5
ЛИСТИНГ 7

ПОСТАНОВКА ЗАДАЧИ

1. На языке C/C++/C# реализовать функцию DGEMM BLAS последовательное умножение двух квадратных матриц с элементами типа double. Обеспечить возможность задавать размерности матриц в качестве аргумента командной строки при запуске программы. Инициализировать начальные значения матриц случайными числами.

2. Провести серию испытаний и построить график зависимости времени выполнения программы от объёма входных данных. Например, для квадратных матриц с числом строк/столбцов 1000, 2000, 3000, … 10000.

3. Оценить предельные размеры матриц, которые можно перемножить на вашем вычислительном устройстве.

4. Реализовать дополнительную функцию DGEMM_opt_1, в которой выполняется оптимизация доступа к памяти, за счет построчного перебора элементов обеих матриц.

5. * Реализовать дополнительную функцию DGEMM_opt_2, в которой выполняется оптимизация доступа к памяти, за счет блочного перебора элементов матриц. Обеспечить возможность задавать блока, в качестве аргументафункции.

6. ** Реализовать дополнительную функцию DGEMM_opt_3, в которой выполняется оптимизация доступа к памяти, за счет векторизации кода.

7. Оценить ускорение умножения для матриц фиксированного размера, например, 1000x1000, 2000x2000, 5000x5000, 10000x10000.

* Для блочного умножения матриц определить размер блока, при котором достигается максимальное ускорение.

8. С помощью профилировщика для исходной программы и каждого способа оптимизации доступа к памяти оценить количество промахов при работе к КЭШ памятью (cache-misses).

9. Подготовить отчет отражающий суть, этапы и результаты проделанной работы.

ОПИСАНИЕ ПРОГРАММЫ

Инструменты, используемые в ходе работы: perf, терминал, текстовый редактор, компилятор C++.

- Максимальный размер матрицы на нашем вычислительном устройстве – 26000x26000
- При данном размере матрицы оперативная память загружается на максимум и также файл подкачки загружен на максимум, при большем размере матрицы система убивает процесс.
- `void DGEMM` – функция поэлементного умножения матрицы.
- `void DGEMM_OPT1` – функция построчного умножения матрицы.
- `void DGEMM_OPT2` – функция блочного умножения матрицы.
- `void rand_mass` – функция заполнения матрицы случайными числами.

РЕЗУЛЬТАТ ПРОГРАММЫ

```
glass@glass-VirtualBox:~/NewLabACS/ABC/lab4$ sudo perf stat -e cache-misses ./heh single opt0 5000

1471.03

 Perfomance counter stats for './heh single opt0 5000':

    27 756 912 632     cache-misses

    1471,184561247 seconds time elapsed

    1471,98399000 seconds user
       0,11799200 seconds sys
```

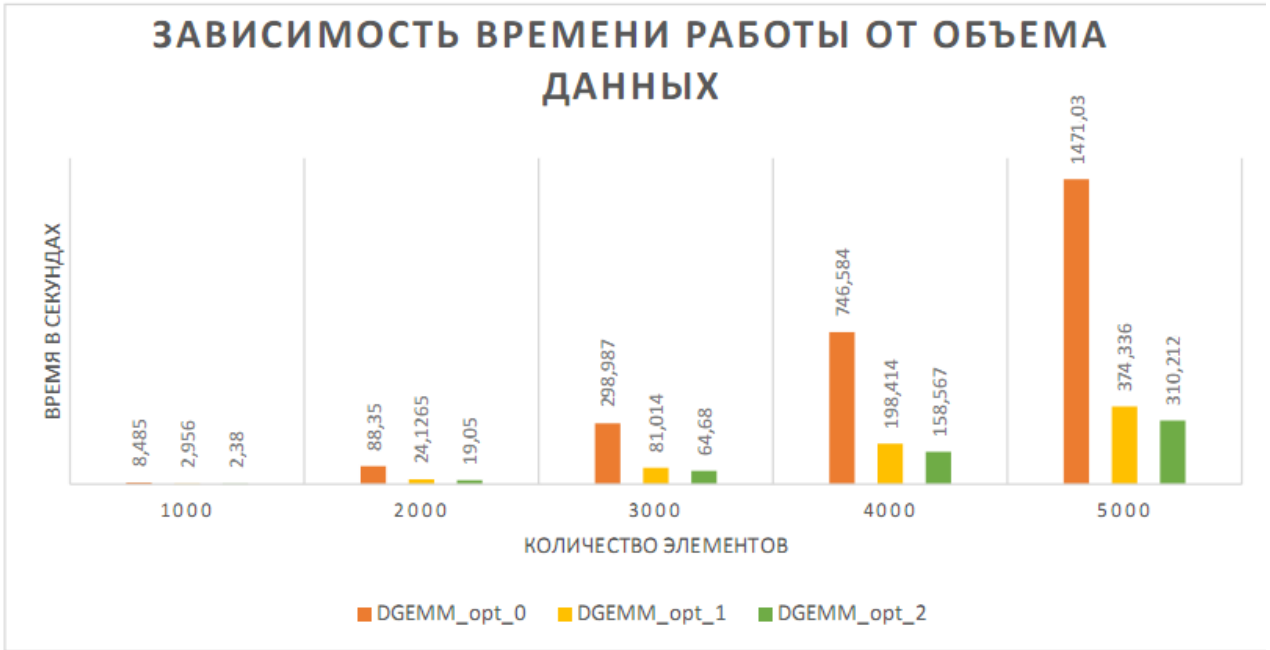
Рисунок 1 Обычный метод перемножения на матрице 5000x5000

```
glass@glass-VirtualBox:~/NewLabACS/ABC/lab4$ sudo perf stat -e cache-misses ./heh single opt1 5000

374.336

        Perfomance counter stats for './heh single opt1 5000':

           24 864 239 521       cache-misses

            374,360632532 seconds time elapsed

            374,241780000 seconds user
              0,157899000 seconds sys
```

Рисунок 2 Построчный метод перемножения на матрице 5000x5000

```
glass@glass-VirtualBox:~/NewLabACS/ABC/lab4$ sudo perf stat -e cache-misses ./heh single opt2 5000

310.212

 Perfomance counter stats for './heh single opt2 5000':

     4 531 384 376     cache-misses

     310,164128342 seconds time elapsed

     310,04671000 seconds user

       0,11499000 seconds sys
```

Рисунок 3 Блочный метод перемножения на матрице 5000x5000



По приложенным гистограммам видно, что самый эффективный способ перемножения матриц это - метод блочного перебора элементов матриц, за ним следует метод построчного перебора элементов матриц, а после уже обычное перемножение. В блочном методе мы использовали размер блока равный 8. Также из гистограммы видно, что промахи при работе с кэш памятью сохраняют свою очередность, как и в скорости, то есть – в блочном методе меньше всего промахов, за ним построчный, а далее идёт обычный метод.

ЛИСТИНГ

```
#include <bits/stdc++.h>

using namespace std;

enum {
    BS = 8
};

void DGEMM(double **a, double **b, double **c, long long N) {
    for (long long i = 0; i < N; i++) {
        for (long long j = 0; j < N; j++) {
            for (long long k = 0; k < N; k++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}

void DGEMM_OPT1(double **a, double **b, double **c, long long N) {
    for (long long i = 0; i < N; i++) {
        for (long long k = 0; k < N; k++) {
            for (long long j = 0; j < N; j++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}

void DGEMM_OPT2(double **a, double **b, double **c, long long N) {
    int i0 = 0, k0 = 0, j0 = 0, str1 = 0, str2 = 0, n = N;
    double *a0 = NULL, *b0 = NULL, *c0 = NULL;

    for (int i = 0; i < n; i += BS) {
        for (int j = 0; j < n; j += BS) {
            for (int k = 0; k < n; k += BS) {
                for (i0 = 0, c0 = &c[i][j], a0 = &a[i][k], str1 = i+1; i0
< BS;++i0, c0 = &c[str1][j], a0 = &a[str1][k], str1+=1){
                    for (k0 = 0, b0 = &b[k][j], str2 = k+1;k0 < BS; ++k0,
b0 = b[str2], str2 += 1) {
                        for (j0 = 0; j0 < BS; ++j0){
                            c0[j0] += a0[k0] * b0[j0];
                        }
                    }
                }
            }
        }
    }
}


void rand_mass(double **a, double **b, long long n) {
    srand (time(NULL));
    for(long long i = 0; i < n; ++i){
        for(long long j = 0; j < n; ++j){
            a[i][j] = rand() % 100;
            b[i][j] = rand() % 100;
        }
```

```
    }
}

void series_func(string multiply, int left, int right) {
    clock_t start, finish;
    if (multiply == "opt0") {
        start = clock();
        ofstream out("result_opt0.txt");
        out << "Тип\tРазмер\tВремя" << endl;
        for (int k = left; k <= right; k += 1000) {
            double **a, **b, **c;
            a = new double *[k];
            b = new double *[k];
            c = new double *[k];

            for (long long i = 0; i < k; i++) {
                a[i] = new double[k];
                b[i] = new double[k];
                c[i] = new double[k];
                for (long long j = 0; j < k; j++) {
                    a[i][j] = 0;
                    b[i][j] = 0;
                    c[i][j] = 0;
                }
            }
            rand_mass(a, b, k);
            DGEMM(a, b, c, k);
            finish = clock();
            cout << endl << (double) (finish - start) / CLOCKS_PER_SEC <<
endl;
            out << multiply << "\t" << k << "\t" << (double) (finish -
start) / CLOCKS_PER_SEC << endl;

            for (int i = 0; i < k; i++) {
                delete[]a[i];
                delete[]b[i];
                delete[]c[i];
            }
            delete[]a;
            delete[]b;
            delete[]c;
        }
    } else if (multiply == "opt1") {
        start = clock();
        ofstream out("result_opt1.txt");
        out << "Тип\tРазмер\tВремя" << endl;
        for (int k = left; k <= right; k += 1000) {
            double **a, **b, **c;
            a = new double *[k];
            b = new double *[k];
            c = new double *[k];

            for (long long i = 0; i < k; i++) {
                a[i] = new double[k];
                b[i] = new double[k];
                c[i] = new double[k];
                for (long long j = 0; j < k; j++) {
                    a[i][j] = 0;
```

```
                    b[i][j] = 0;
                    c[i][j] = 0;
                }
            }
            rand_mass(a, b, k);
            DGEMM_OPT1(a, b, c, k);
            finish = clock();
            cout << endl << (double) (finish - start) / CLOCKS_PER_SEC <<
endl;
            out << multiply << "\t" << k << "\t" << (double) (finish -
start) / CLOCKS_PER_SEC << endl;
            for (int i = 0; i < k; i++) {
                delete[]a[i];
                delete[]b[i];
                delete[]c[i];
            }
            delete[]a;
            delete[]b;
            delete[]c;
        }
    } else if (multiply == "opt2") {
        start = clock();
        ofstream out("result_opt2.txt");
        out << "Тип\tРазмер\tВремя" << endl;
        for (int k = left; k <= right; k += 1000) {
            double **a, **b, **c;
            a = new double *[k];
            b = new double *[k];
            c = new double *[k];

            for (long long i = 0; i < k; i++) {
                a[i] = new double[k];
                b[i] = new double[k];
                c[i] = new double[k];
                for (long long j = 0; j < k; j++) {
                    a[i][j] = 0;
                    b[i][j] = 0;
                    c[i][j] = 0;
                }
            }
            rand_mass(a, b, k);
            DGEMM_OPT1(a, b, c, k);
            finish = clock();
            cout << endl << (double) (finish - start) / CLOCKS_PER_SEC <<
endl;
            out << multiply << "\t" << k << "\t" << (double) (finish -
start) / CLOCKS_PER_SEC << endl;
            for (int i = 0; i < k; i++) {
                delete[]a[i];
                delete[]b[i];
                delete[]c[i];
            }
            delete[]a;
            delete[]b;
            delete[]c;
        }
    }
}
```

```
void single_func(string multiply, int n) {
    if (multiply == "opt0") {
        clock_t start, finish;
        start = clock();
        double **a, **b, **c;
        a = new double *[n];
        b = new double *[n];
        c = new double *[n];

        for (long long i = 0; i < n; i++) {
            a[i] = new double[n];
            b[i] = new double[n];
            c[i] = new double[n];
            for (long long j = 0; j < n; j++) {
                a[i][j] = 0;
                b[i][j] = 0;
                c[i][j] = 0;
            }
        }

        rand_mass(a, b, n);
        DGEMM(a, b, c, n);
        finish = clock();
        cout << endl << (double) (finish - start) / CLOCKS_PER_SEC <<
endl;
    } else if (multiply == "opt1") {
        clock_t start, finish;
        start = clock();
        double **a, **b, **c;
        a = new double *[n];
        b = new double *[n];
        c = new double *[n];

        for (long long i = 0; i < n; i++) {
            a[i] = new double[n];
            b[i] = new double[n];
            c[i] = new double[n];
            for (long long j = 0; j < n; j++) {
                a[i][j] = 0;
                b[i][j] = 0;
                c[i][j] = 0;
            }
        }

        rand_mass(a, b, n);
        DGEMM_OPT1(a, b, c, n);
        finish = clock();
        cout << endl << (double) (finish - start) / CLOCKS_PER_SEC <<
endl;
    } else if (multiply == "opt2") {
        clock_t start, finish;
        start = clock();
        double **a, **b, **c;
        a = new double *[n];
        b = new double *[n];
        c = new double *[n];
```

```
        for (long long i = 0; i < n; i++) {
            a[i] = new double[n];
            b[i] = new double[n];
            c[i] = new double[n];
            for (long long j = 0; j < n; j++) {
                a[i][j] = 0;
                b[i][j] = 0;
                c[i][j] = 0;
            }
        }

        rand_mass(a, b, n);
        DGEMM_OPT2(a, b, c, n);
        finish = clock();
        cout << endl << (double) (finish - start) / CLOCKS_PER_SEC <<
endl;
    }
}

int main(int argc, char** argv) {
    if(argc < 3){
        cout << "Слишком мало аргументов";
        return 1;
    } else if((string)argv[1] == "single"){
        single_func((string)argv[2], atoi(argv[3]));
    } else if((string)argv[1] == "series") {
        if (argc < 5) {
            cout << "Слишком мало аргуметов для серии";
            return 1;
        } else{
            series_func((string) argv[2], atoi(argv[3]), atoi(argv[4]));
        }
    }
}
```